EAvatar ME Documentation

Release 0.1.0

EAvatar Technology Ltd.

November 13, 2015

Contents

Contents:

CHAPTER 1

Introduction

EAvatar ME, or Avame for short, is an event-driven agent for task automation. It's designed with following scenarios in mind:

- Web scraping
- Web functional tests
- Network service monitoring
- Cloud application integration

Basically speaking, Avame runs tasks on behalf of the user in the background.

A task is a unit of work that performs one action only. Related tasks are grouped as a job for complicated workflow. Incidentally, a job is the unit for submission and is described by a script. The scripts are nothing but restrictive Python code. Compared to regular Python code, a script do not support following features, just to name a few:

1. No import statement
2. No while loop
3. No print statement
4. Names start with double underscores are prohibited.
5. No function definition
6. No class definition

All these restrictions are to make scripts easier to write and read. Other heavy-lifting actions are provided by modules and exposed to scripts.

The source code of Avame project is released with Apache license 2.0

CHAPTER 2

User Guide

Avame not yet released, the document is for your information only.

2.1 System Requirements

2.1.1 Supported Environments

Avame is a cross-platform application which supports following desktop environments:

1. Windows 7 or above (X86-64 architecture)
2. OS X 10.8+ (X86-64 architecture)
3. Ubuntu 14.04+ Unity desktop (X86-64 architecture)

2.1.2 Web Browsers

Avame uses your default web browser to access the web user interface (a.k.a. webfront).

- Google Chrome 46.0+
- Firefox 41.0+
- Internet Explorer 11

2.2 Installation

For desktop environments, such as Windows, OS X, etc, Avame is distributed as a portable application. Following are the usual steps to install Avame for these environments:

1. **Download the distribution package for your environment.** The package might be in different formats for various environments.
   - avame-x.y.x.zip for Windows
   - avame-x.y.z.tgz for Ubuntu
   - avame-x.y.z.dmg for OS X

   , where 'x.y.z' is the release version.

2. **Uncompress the package to a suitable directory.** Use the existing tool to uncompress the distribution package.
3. **Delete the downloaded distribution package.** This step is optional.

2.2.1 OS X

Avame is released as an application bundle within a disk image (.dmg). Follow the steps to install Avame on OS X:

1. Double click the disk image to mount it.
2. All you have to do is copy the application bundle to your Applications folder.
3. Unmount the disk image.

2.3 Launch the Application

Assume APPFOLDER is the installation folder of Avame application. You may launch Avame via command line or File manager or the like.

2.3.1 Windows

Launch the application from command line.

```
APPFOLDER\avame.exe
```

2.3.2 OS X

Launch from the console

```
APPFOLDER/avame
```

or run the app bundle:

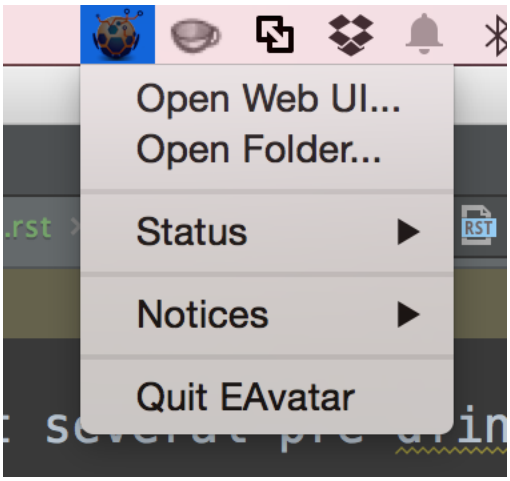
2.3.3 Ubuntu

Launch from the console

```
APPFOLDER/avame
```

2.4 Application System Tray

As an agent, Avame works silently most of the time. The main UI is a system tray icon on the desktop environment of your choice. Through the tray icon, you may trigger a context menu illustrated as follow:



2.5 Open Web UI (Not working yet)

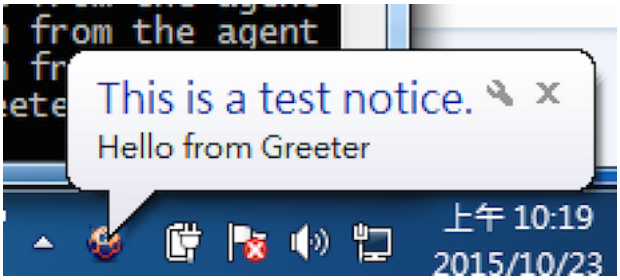
From the context menu, choose the 'Open Web UI...' option or input 'http://127.0.0.1:5080/' in your browser's address bar. Following figure illustrates the home page of the web UI (Fake):

2.6 Get Notified

Avame may notify you from time to time. The way how it notifies is platform-specific, though.

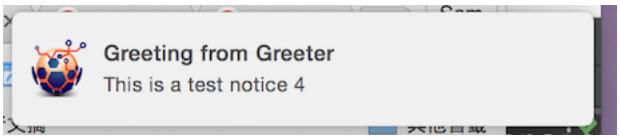
2.6.1 For Windows

System tray notification is used.

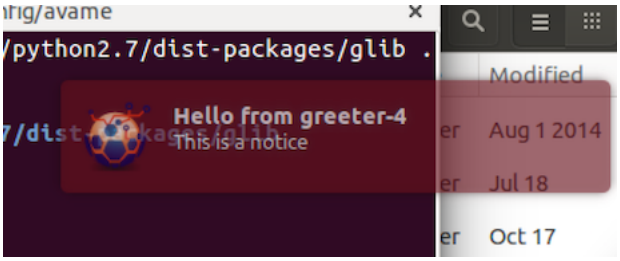


2.6.2 For OS X

Avame supports the notification center.

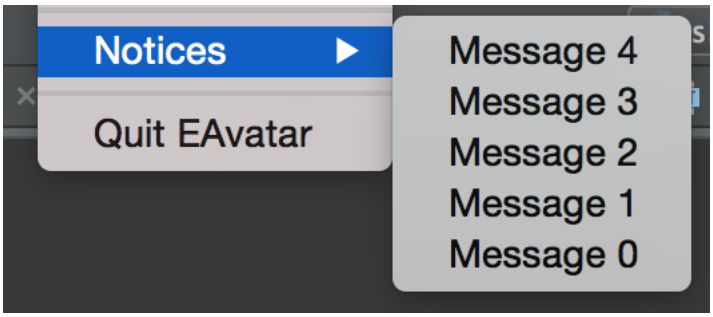


2.6.3 For Ubuntu

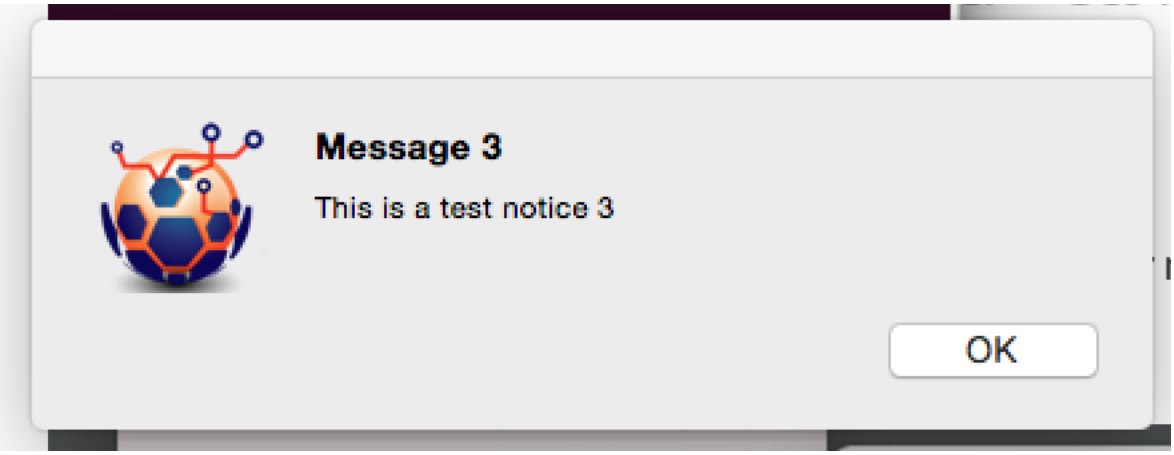


2.7 Recent Notices

In case that you missed some notifications from Avame, fear not. 10 most recent notices are kept in the context menu for your convenience:



Choose the notice you want to read, a message box should show up:



2.8 Application Folder

2.8.1 Location

Avame creates a per-user application folder when runs for the first time. The location is dependent on the operating system, following are the typical paths:

- **Mac OS X:** ~/Library/Application Support/avame
- **Mac OS X (POSIX):** ~/.avame
- **Ubuntu:** ~/.config/avame
- **Win 7 (roaming):** C:\Users\<user>\AppData\Roaming\avame
- **Win 7 (not roaming):** C:\Users\<user>\AppData\Local\avame

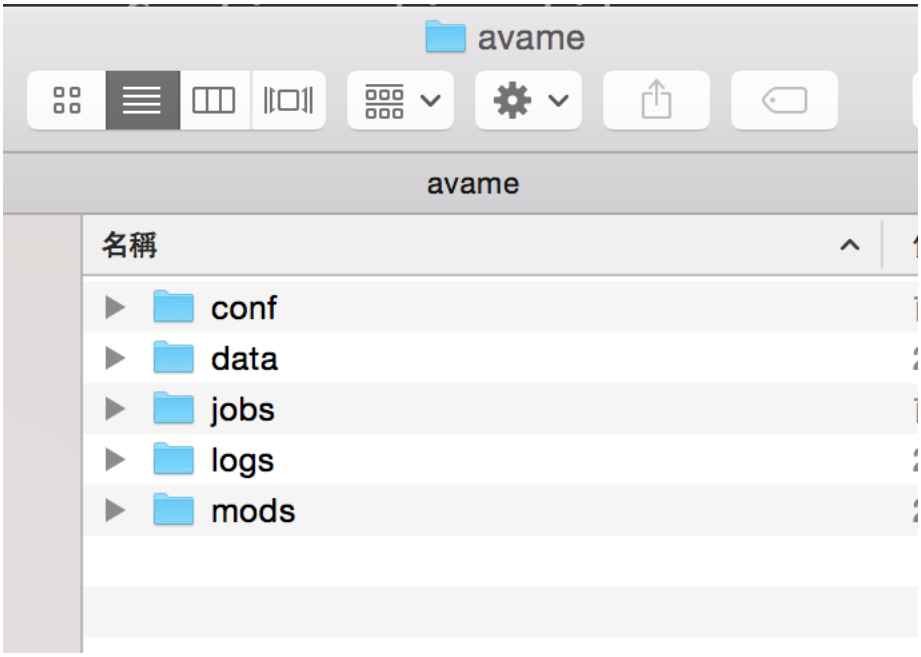
2.8.2 Structure

Under the application folder, there exist several pre-defined subfolders. Users may create more for other purposes.

1. **conf/** Configuration files.
2. **data/** Folder for storing generic data.
3. **jobs/** Auto-start jobs.
4. **logs/** Log files.
5. **mods/** Standalone module files. Modules in this folder will be imported when start.
6. **pkgs/** Python package distributions in EGG format, some of them may also be Avame's extensions.

2.8.3 Open Folder

From the context menu, choose the 'Open Folder...' option to open Avame application folder.


avame
avame
名稱
conf
data
jobs
logs
mods

CHAPTER 3

Writing Scripts

3.1 What are Scripts

To tell Avame to do your jobs, you need to tell Avame how to do it imperatively. The instructions are expressed as a script in Python-like programming language. It sounds scary at the first place to writing scripts in a programming language for regular users. It ends up not so hard at all.

3.2 Differences from Regular Code

The syntax is intentionally very limited so that it's more approachable than full-featured Python codes. The document is not intended to describe all the syntax as it's a proper subset of Python's. Compared to regular Python, following are removed features:

1. No import statement
2. No while control loop
3. No print statement
4. Names start with double underscores are prohibited, e.g. '__class__'.
5. No function definition
6. No class definition

3.3 Common Libraries

It's not supported to import modules for scripts, so some standard libraries from Python runtime are selected and made available. The modules provided:

1. datetime
2. collections
3. calendar
4. heapq
5. bisect
6. array
7. queue or Queue

8. string
9. re
10. math
11. random
12. json
13. base64
14. binascii
15. hashlib
16. hmac

In addition to standard libraries, *lxml* is available for XML/HTML parsing.

3.4 Perform Actions

Besides the common libraries, scripts can perform various actions, each of which are explicitly registered functions and are intended for use by scripts.

The syntax to invoke an action is like follow:

```
ava.do(action_name, **kwargs)
```

, where *action_name* is in a format like 'mod_name.func_name'. For example, 'imap.check_gmail' is the action name from *imap* module to check GMail.

ava.do function returns a task object whose essential methods are:

1. **result(blocked=True, timeout=None)** Wait for the task to finish and return the result (or raise an exception). By default, the method blocks the caller. But can be made to work asynchronously. If *blocked* is False and the task is not stopped, a *Timeout* exception is raised. If *timeout* parameter is given, a *Timeout* exception will be raised if expired.
2. **stopped()** Check if the task is finished or failed without blocking.
3. **finished()** Check if the task is completed successfully.
4. **failed()** Check if the task failed.

The execution of an action is represented as a task. Multiple tasks can be issued concurrently. At times, it's needed to wait for tasks to finish. To coordinate tasks, a script may use following method to wait for tasks.

```
ava.wait(tasks, timeout=10)
```

,where *tasks* is a list of task objects; *timeout* is a timeout interval in seconds. There is an additional argument *count* which specify how many tasks to wait for before returning.

3.5 Built-in Actions

3.5.1 User Module

- **user.notify** Notify owner with a message and title.

3.5.2 Request Module

Make methods from *requests* library available as actions.

- requests.head
- requests.get
- requests.put
- requests.patch
- requests.delete
- requests.post

3.6 Loop Control

The only supported loop control for scripts is *for* statement. *for* loop is usually used to iterate a finite number of elements. In case a indefinite loop is needed, following construct can be used:

```
for it in ava.schedule:
    ...
```

ava.schedule is a special generator that returns counting number from 1 for each iteration. By default, the interval between intervals are 1 minute. Following are more examples:

3.6.1 1-minute interval

```
for it in ava.schedule.every(1).minute
```

3.6.2 5-minute interval

```
for it in ava.schedule.every(5).minutes
```

There are other supported interval units like *second*,'seconds', *hour*, *hours*, *day* and *days*.

The total number of looping can also be control by providing a *counts* value. For example:

```
for it in ava.schedule.count(5).every().minute:
    ...
```

Above code snippet loops 5 times and waits for 1 minute before an iteration.

3.7 Example

Following script is for checking a GMail account with IMAP protocol every minute:

```
last_unseen = 0

for it in ava.schedule.every(1).minute:
    check_task = ava.do('imap.check_gmail',
                        username='username@gmail.com',
                        password='password')
```

```
    messages, unseen = check_task.result()
    if unseen > 0 and unseen != last_unseen:
        last_unseen = unseen
        ava.do('user.notify',
               message="You got %d new messages." % unseen,
               title="You Got Mails from GMail")
```

CHAPTER 4

Development

For developers who want to build Avame from source code, proper development environments should be set up. According to the platform, the system requirements and instructions are different.

Avame's source code is hosted on GitHub at https://github.com/eavatar/eavatar-me . Therefore, Git is used for management of the source code.

virtualenv is used to have a relatively isolated Python runtime during development. However, it becomes tricky due to the fact that Avame needs access to some packages that cannot be installed via *pip*. So, even *virtualenv* is used, the *–system-site-packages* argument is set for all supported platforms.

Refer to the section for the platform of your choice. Should you encountered any issues , please file issues at https://github.com/eavatar/eavatar-me/issues

4.1 On Windows Platform

4.1.1 System Requirements

Operating system	CPU Architecture
Windows 7 Pro	X86-64

4.1.2 Git for Windows

To check out source codes on Windows platform, *Git for Windows* command line tool is needed. Please download and install it from https://git-scm.com/download/win .

Please note that all command-line instructions are assumed to be run using the shell provided by Git for Windows, which provides a Unix-like interface.

4.1.3 Python Distribution

The Python distribution for development on Windows platform is Anaconda Python 2.7. Download it from https://www.continuum.io/downloads .

4.1.4 Get the Source Code

Use following command to check out the source code:

```
git checkout https://github.com/eavatar/eavatar-me avame
```

It's assumed that source code is cloned to the local host at $WORKDIR directory.

4.1.5 Virtual Environment

```
virtualenv --system-site-packages env
source env/Scripts/activate
```

It's assumed that command-line instructions in the following sections are run with the virtual environment activated.

4.1.6 Installing Dependencies

Use following instruction to install

```
pip install -r requirements/requirements_win32.txt
```

Avame needs 'pywin32', 'lxml' packages on Windows platform, which should be already provided by Anaconda Python Distribution.

4.1.7 Run from the Source

To run the program form the source, please ensure following paths are in the *sys.path*. This can be done by setting an environment variable *PYTHONPATH* to be like this:

```
export PYTHONPATH="./src"
```

Note that this is assumed to be run from the *Git for Windows* shell. If you use an IDE, the root and the 'src' sub-folder of the project should be in the paths as mentioned above.

4.1.8 Build the Binaries

```
pyinstaller pack\avame.spec --clean -y
```

The application binaries are located at $WORKDIRdistavame. You may use following instruction to launch it:

```
.\dist\avame\avame.exe
```

4.2 On OS X Platform

4.2.1 System Requirements

The instructions are tested on following configuration.

Operating system	CPU Architecture
OS X 10.10	X86-64

4.2.2 Python Distribution

Although OS X has a bundled Python runtime, the one used should be from python.org. Please download version 2.7.10 from that site.

Note that it's assumed you have XCode command-line tools installed already.

4.2.3 Virtual Environment

```
virtualenv --system-site-packages env
source env/bin/activate
```

It's assumed that command-line instructions in the following sections are run with the virtual environment activated.

4.2.4 Get the Source Code

Avame's source code is hosted on GitHub at https://github.com/eavatar/eavatar-me . Therefore, Git is used for management of the source code.

Use following command to check out the source code:

```
git checkout https://github.com/eavatar/eavatar-me avame
```

It's assumed that source code is cloned to the local host at $WORKDIR directory.

4.2.5 Installing Dependencies

On OS X development machine, should you encountered an error with message like *error: '_Noreturn' keyword must precede function declarator*, then *gevent* package needs to be installed with following instruction:

```
CFLAGS='-std=c99' pip install gevent==1.0.2
```

Install other dependencies:

```
pip install -r requirements/requirements_osx.txt
```

lxml package needs extra steps to install with following instructions:

```
brew install libxml2
pip install lxml
```

4.2.6 Run from the Source

To run the program form the source, please ensure following paths are in the *sys.path*. This can be done by setting an environment variable *PYTHONPATH* to be like this:

```
export PYTHONPATH="./src"
```

If you use an IDE, the the 'src' subfolder of the project should be in the paths as mentioned above.

4.2.7 Build the Binaries

```
pyinstaller pack/avame.spec --clean -y
```

The application binaries are located at $WORKDIR/dist/avame/. You may use following command to launch it:

```
./dist/avame/avame
```

In addition to the binaries, an application bundle for OS X is created and placed at $WORKDIR/dist/EAvatar.app/ . An application bundle is a special directory on OS X, which may be launched from the command line with following instruction:

```
open ./dist/EAvatar.app
```

4.3 On Ubuntu Platform

4.3.1 System Requirements

Operating system	CPU Architecture
Ubuntu 14.04	X86-64

4.3.2 Python Distribution

Avame uses the system-bundled Python distribution for the development on Ubuntu platform. Note that the version used is 2.7.6.

4.3.3 Virtual Environment

```
virtualenv --system-site-packages env
source env/bin/activate
```

It's assumed that command-line instructions in the following sections are run with the virtual environment activated.

4.3.4 Get the Source Code

Avame's source code is hosted on GitHub at https://github.com/eavatar/eavatar-me . Therefore, Git is used for management of the source code.

Use following command to check out the source code:

```
git checkout https://github.com/eavatar/eavatar-me avame
```

It's assumed that source code is cloned to the local host at $WORKDIR directory.

4.3.5 Installing Dependencies

Use following instructions to install Python and system packages:

```
sudo apt-get install libxml2-dev libxslt1-dev python-dev python-lxml
pip install -r requirements/requirements_gtk.txt
```

4.3.6 Run from the Source

To run the program form the source, please ensure following paths are in the *sys.path*. This can be done by setting an environment variable *PYTHONPATH* to be like this:

```
export PYTHONPATH="./src"
```

If you use an IDE, the root and the 'src' subfolder of the project should be in the paths as mentioned above.

4.3.7 Build the Binaries

```
pyinstaller pack/avame.spec --clean -y
```

The application binaries are located at $WORKDIR/dist/avame/. You may use following command to launch it:

```
./dist/avame/avame
```

CHAPTER 5

Frequently Asked Questions

5.1 Where is my working folder for Avame?

Avame creates a per-user application folder when runs for the first time. The location is dependent on the operating system, following are the typical paths:

- **Mac OS X:** ~/Library/Application Support/avame
- **Mac OS X (POSIX):** ~/.avame
- **Ubuntu:** ~/.config/avame
- **Win 7 (roaming):** C:\Users\<user>\AppData\Roaming\avame
- **Win 7 (not roaming):** C:\Users\<user>\AppData\Local\avame

5.2 How do I add auto-run jobs when Avame starts?

Simply add the script file (with extension .py) to the subfolder 'jobs' under the Avame's working folder, and then restart Avame. It will be started if it's a correct script.

CHAPTER 6

Internals

6.1 Event-driven, Non-blocking I/O

At the core of EAvatar, Gevent is used as the event-driven, non-blocking I/O engine. With the lightweight concurrency support and thread-like programming model, it a pleasure again to write asynchronous I/O codes.

6.2 Restrictive Execution of Python Codes

In addition to mainly developed in Python, EAvatar also uses Python as the domain-specific language(DSL) for describing jobs. A job is defined by a script in Python with very limited subset of features enabled.

A script is parsed into an abstract syntax tree with *ast* module, and then being walked through to detect invalid constructs such as 'import', 'print', 'functiondef', etc. Without complex constructs, it makes writing scripts more approachable for average users.

6.3 How Avame integrated with desktop environments

The main user interface for Avame is a so-called system tray icon or status icon on the desktop environment. Trying to avoid the bloated cross-platform GUI frameworks like QT, it takes a very different way to provide the desktop UI.

No cross-platform GUI frameworks are used for the desktop environments, an environment-by-environment integration is used instead.

- On Windows platform, Win32 API is used via *PyWin32* package.
- On OS X platform, Cocoa is used via *PyObjc* package.
- On Ubuntu platform, GTK3 is used via *PyGObject* package.

A cross-platform web-based user interface is supported for user interactions that are more complex. To this end, Avame launches the default web browser when web UI is required.

CHAPTER 7

Glossary

The project uses terms that are overloaded. It's useful to highlight some commonly used terms and definitions.

Job A set of related tasks executed in an order determined by a script.

Task A unit of work in a job, which also represents the execution of an action.

Action A function that can be invoked by jobs. Each invocation of an action is represented as a task.

Script The code to describe the flow in which tasks should be executed.

Module A software component providing functions, classes and other definitions.

Package A software component which contains sub-packages or modules.